INF8801 - Inpainting

Hiver 2025

1 Description globale

1.1 But

Le but de ce TP est d'implémenter la méthode appelée *Scene Completion* décrite dans l'article suivant : `http://graphics.cs.cmu.edu/projects/scene-completion/`. Il s'agit d'une méthode permettant de combler des trous dans une image (inpainting), en utilisant le morceau d'image le plus *adapté* trouvé dans une très grande base d'images. Ce TP est divisé en deux parties indépendantes : la première consiste en la recherche d'images similaires dans une base; la seconde consiste en l'inpainting à proprement dit: la composition des deux images.

1.2 Partie de la méthode fournie

Un canevas vous est fourni, et vous n'aurez à compléter que certaines étapes du code. L'algorithme global de *Scene Completion* (y compris les parties que vous n'aurez pas à implémenter) se décompose comme ceci :

- Calcul du descripteur GIST sur la base de données (**Section 2**)
- Requête des images les plus proches dans la base (**Section 2**)
- Translation optimale
- Calcul du découpage optimal avec GraphCut
- Composition avec l'algorithme de Poisson (**Section 3**)

2 Recherche dans la base de données

Le descripteur VLAD (basé sur SIFT, ORB ou similaire) vu au TP1 n'est plus adapté ici. En effet, celui-ci cherchait à énoncer et décrire de manière robuste les éléments spécifiques

contenus dans l'image. Seulement, ce n'est plus adapté car il faut à présent décrire le contenu de l'image dans son ensemble, l'idée générale, d'où le nom "GIST".

Ce descripteur est construit à l'aide de filtres de Gabor

2.1 Filtres de Gabor

Dans cette partie, nous allons étudier le descripteur utilisé sur cette base de données. Il a deux composantes : la première décrit les couleurs de l'image, la seconde décrit les contours de l'image : il s'agit de *GIST*.

Pour obtenir les contours, on doit d'abord définir un ensemble de filtres de Gabor. Un filtre de Gabor est intégralement décrit par les quatre paramètres σ_x, σ_y, λ et θ selon l'équation 2.1. Ces filtres sont encore parfois utilisés pour extraire des features de haut niveau dans des architectures d'apprentissage profond.

$$\begin{aligned} G_{\{\sigma_x,\sigma_y,\lambda,\theta\}}(x,y) &= \frac{1}{2\pi\sigma_x\sigma_y}\exp\left(\frac{-x'^2}{2\sigma_x^2}\right)\exp\left(\frac{-y'^2}{2\sigma_y^2}\right)\sin\left(2\pi\frac{x'}{\lambda}\right) \\ \text{où } &\begin{cases} x' = x\cos(\theta) - y\sin(\theta) \\ y' = x\sin(\theta) + y\cos(\theta) \end{cases} \end{aligned} \tag{1}$$

Pour décrire efficacement les contours à différentes échelles et orientations, on définit un ensemble de filtres de Gabor pour 6 orientations et 5 valeurs d'échelle (30 filtres au total). Les orientations sont comptées entre 0 inclus et π exclus car le sens n'a pas d'influence (pourquoi?).

Remarque : dans ce TP, on considère simplement que pour une valeur d'échelle s donnée on utlise $\lambda = \sigma_y = s$ et $\sigma_x = s/2$.

2.2 Descripteur GIST

Pour cette question, vous devez compléter la classe *DescGIST*. En particulier, vous devez implémenter les fonctions:

- `DescGIST.get_gabor()` qui renvoie les filtres de Gabor pour un certain nombres de paramètres spécifiés
- `DescGIST.get_all_gabors()` qui calcule tous les filtres de Gabor pour un certain nombre d'orientations (self.nOrientations = 6) et de facteurs d'échelle (self.nScales = 5). Nous obtiendrons ainsi 30 filtres c.f. figure 1. On veut des facteurs d'échelle égaux à la taille du filtre divisée par 2, 4, 8, etc.
- `DescGIST.describe()` qui calcule la réponse d'une image à un jeu de filtres de Gabor (self.filters) et enregistre cette réponse dans self.values. L'image en entrée doit être

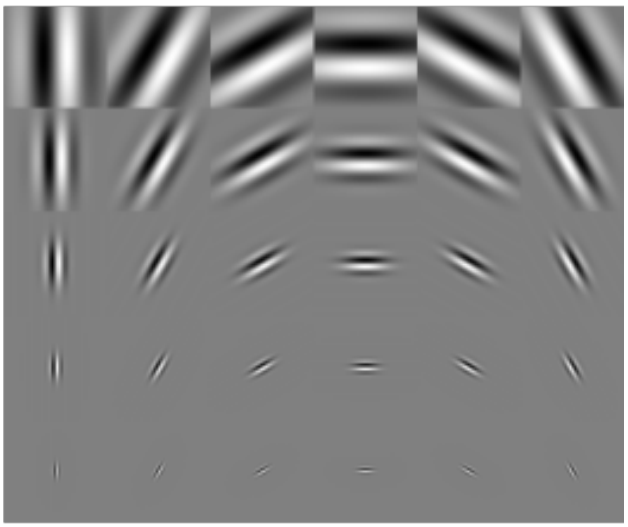

Figure 1: Filtres de Gabor que vous devriez obtenir

convertie en niveaux de gris et redimensionnée. La valeur absolue de la réponse de chaque filtre doit ensuite être moyennée sur les régions de l'image correspondant à une grille de taille 4x4 (un simple cv2.resize() suffit). Ces valeurs peuvent être affichées sous forme de graphique avec la fonction `DescGIST.display()` qui vous est donnée (c.f. figure 2).

Image en niveaux de gris | Réponse à un filtre horizontal | Descripteur Gist

Figure 2: Exemples de réponses aux filtres de Gabor. Le graphique à droite représente la réponse de chacune des 4x4 régions de l'image aux filtres. Vous pouvez obtenir ce graphique avec la fonction DescGIST.display(). Chaque rosace correspond au set de filtres de Gabor (à différentes échelles et orientations).

Explication de la visualisation du descripteur GIST: La réponse d'une image à un filtre de Gabor est une image de taille 4x4, et nous avons 30 filtres. Visualiser les 30 filtres côte à côte ne serait pas du tout optimal afin de comparer la réponse de notre image à nos différents filtres. La fonction `DescGIST.display()` fonctionne de la manière suivante: Pour chaque pixel de notre image 4x4, on affiche les 30 valeurs associées à chaque filtre sur un disque. Ce disque est divisé en 30 régions: une région par filtre.

3 Seamless carving

Dans cette partie, une image similaire à l'image de requête vous est donnée. Vous devez assembler les deux images afin de combler au mieux le trou dans l'image requête.

3.1 Algorithme d'édition de Poisson

Il s'agit de la dernière étape de l'algorithme. Vous devez implémenter la fonction *poissonBlending* qui prend en entrée 3 arguments :

- L'image à combler (**source**)
- L'image avec laquelle la combler (**cible**)
- Un masque représentant les pixels à combler

Vous utiliserez pour cela l'algorithme de Poisson, décrit dans cet article : *Poisson Image Editing [Perez et al. 2003]*. Le but de cette méthode est de coller l'image dans l'espace du Gradient, puis de l'intégrer pour obtenir le résultat du collage; cette intégration peut être approximée par la méthode de Poisson, en résolvant le problème

$$\triangle I = \nabla G \tag{2}$$

où ∇G est le gradient de l'image source et $\triangle I$ est le Laplacien de la cible.

On ne vous demande pas ici de résoudre cette équation mathématiquement, mais plutôt d'implémenter un algorithme itératif supposé converger vers une solution à cette équation.

Une implémentation possible est d'utiliser la méthode de Jacobi. En effet, résoudre un Laplacien nul sur un ensemble de pixels revient à itérativement assigner à chaque pixel (dans la zone non fixée) la valeur moyenne de ses quatre voisins plus la valeur du Laplacien de la cible.

Afficher dans un même graphique l'image à compléter (la source), l'image à coller (le target) et le résultat, ainsi que des résultats intermédiaires (ex: après 250 itérations, 500, 750, etc.). Le nombre d'itérations choisi doit être clairement affiché et la raison de l'arrêt justifiée (cf. figure 3 où l'utilisateur s'est arbitrairement arrêté à 1000 itérations).

4 Fichiers fournis

Nom fichiers	**Description**
`tp2_name1_name2.ipynb`	Jupyter Notebook à compléter
data/	Images de test

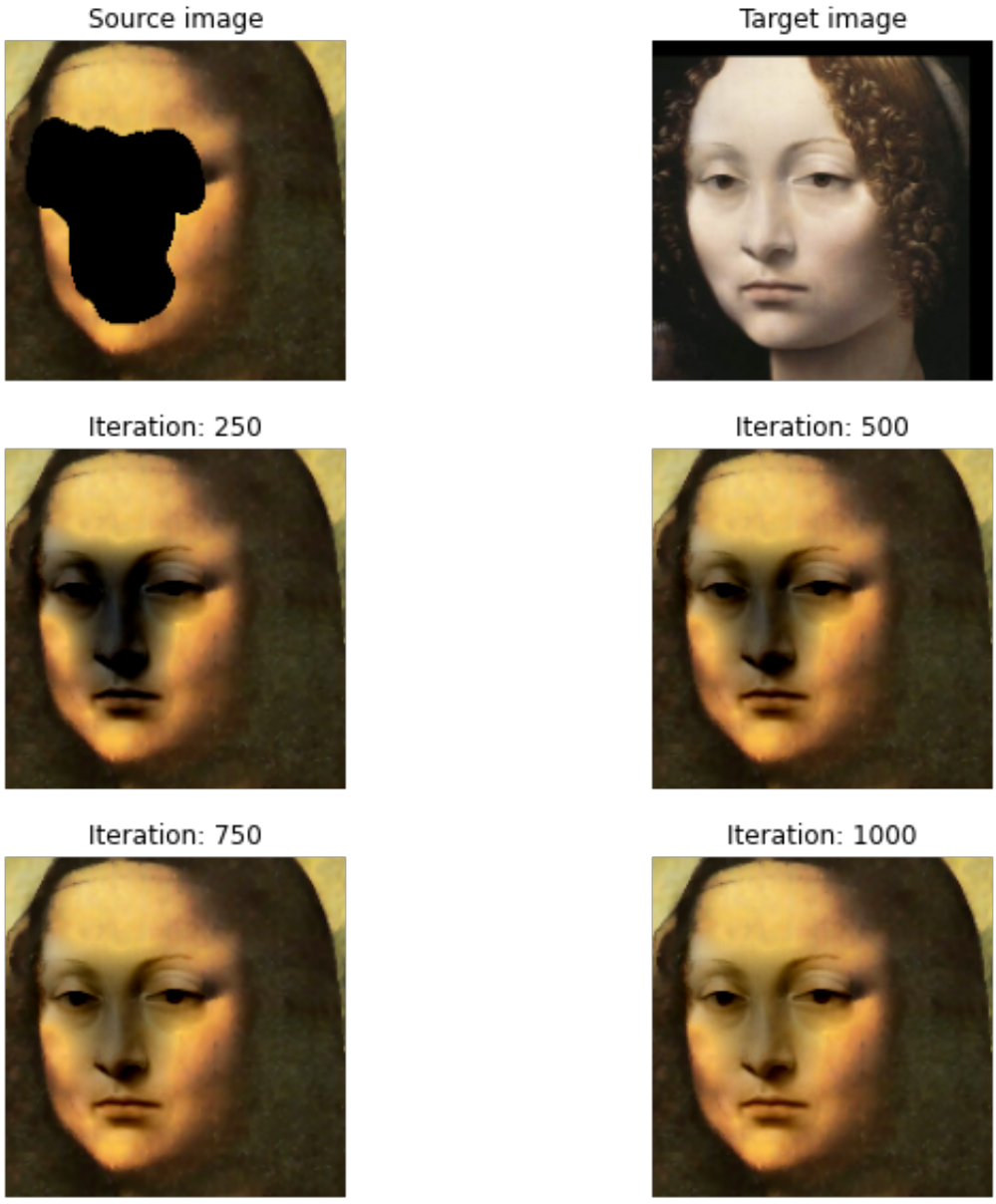

Figure 3: Évolution de la diffusion par le Laplacien (Méthode de Poisson)

5 Exigences

Il vous est demandé de soumettre le Notebook Jupyter avant le **17 Février à 23h59**. Toutes les figures doivent apparaître, et nous devrions être en mesure d'exécuter toutes les cellules dans l'ordre. Vous devez répondre aux questions écrites à l'intérieur du notebook, et votre code doit être **lisible, clair et commenté**.

Les noms des deux élèves doivent apparaître dans le nom du Notebook.